Question 1 - Application of reduction rules for MINIJAVA

Consider the following MINIJAVA classes:

class A {
    int f;
}
class B {
    int f1;
    A f2;
    B m(int x, A a) { this.f1=x; this.f2=a; this; }
}

Evaluate the following expression, starting from an empty store, in the program context consisting of the two classes:

B b = new B.m(3, new A); b.f2.f=5;

Show the names of the rules you use at each step.
Suggestion: write each step on a single line and use names for the stores which appear in steps, defining them separately.

Question 2 - Definition of new reduction rules

Assume we extend MINIJAVA with a conditional operator, with the following syntax:

\[
e ::= \text{if} \left( e_1 \right) \text{else} \left( e_2 \right)
\]

and the standard meaning.

Add reduction rules for the conditional operator (you should add three rules, one for propagation).
What should happen to expressions such as \text{if} \left( 1 \right) \text{else} \left( 2 \right)?

Question 3 - Implementation of recursive data structures (continued)

Consider the following Java implementation of binary trees with integer labels (we omit for simplicity handling of error situations by exceptions).

abstract class BTree {
    abstract boolean isEmpty();
    abstract int label();
    abstract BTree left();
    abstract BTree right();
}

class Node extends BTree {
    private int label;
    private BTree left, right;
    Node(int label, BTree left, BTree right) {
        this.label = label; this.left = left; this.right = right;
    }
    boolean isEmpty() { return false; }
    int label() { return label; }
    BTree left() { return left; }
    BTree right() { return right; }
}
class Empty extends BTree {
  boolean isEmpty () { return true; }
  int label () { //error }
  BTree left () { //error }
  BTree right () { //error }
}

Be careful: do not confuse an empty tree (modeled by an instance of the class Empty) with null.

1. add in BTree a non-abstract method which returns the height of a tree (length of a maximal path, 0 if empty);
2. add in BTree an abstract method which returns the number of nodes of a tree, and implement the method in the subclasses;
3. add a heir class Leaf whose instances represent leaves;
4. define a method (be careful: where and of which kind?) which given two integers $n$ and $m$ returns a binary tree (preferably with minimal height) having as (distinct) labels numbers from $n$ to $m$ (hence empty if $n > m$).